**Data Structure**

**&**

**Algorithm**

**Class 8**

**Lab 13**

|  |
| --- |
| Lab Objectives:DequeData TypePalindromeList |

# What Is a Deque?

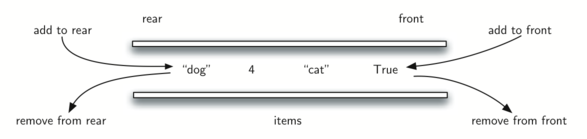
## A deque, also known as a double-ended queue, is an ordered collection of items similar to the queue.



## It has two ends, a front and a rear, and the items remain positioned in the collection.New items can be added at either the front or the rear.

## IMG_256It is important to note that even though the deque can assume many of the characteristics of stacks and queues, it does not require the LIFO and FIFO orderings that are enforced by those data structures.

## It is up to you to make consistent use of the addition and removal operations.



## 

# The Deque Abstract Data Type

## The deque abstract data type is defined by the following structure and operations.The deque operations are given below.

## Deque() creates a new deque that is empty. It needs no parameters and returns an empty deque.

## *addFront(item)* adds a new item to the front of the deque. It needs the item and returns nothing.

## addRear(item) adds a new item to the rear of the deque. It needs the item and returns nothing.

## removeFront() removes the front item from the deque. It needs no parameters and returns the item. The deque is modified.

## removeRear() removes the rear item from the deque. It needs no parameters and returns the item. The deque is modified.

## isEmpty() tests to see whether the deque is empty. It needs no parameters and returns a boolean value.

## size() returns the number of items in the deque. It needs no parameters and returns an integer.

## if we assume that d is a deque then the operations will be like below

|  | | |
| --- | --- | --- |
| **Deque Operation** | **Deque Contents** | **Return Value** |
| d.isEmpty() | [] | True |
| d.addRear(4) | [4] |  |
| d.addRear('dog') | ['dog',4,] |  |
| d.addFront('cat') | ['dog',4,'cat'] |  |
| d.addFront(True) | ['dog',4,'cat',True] |  |
| d.size() | ['dog',4,'cat',True] | 4 |
| d.isEmpty() | ['dog',4,'cat',True] | False |
| d.addRear(8.4) | [8.4,'dog',4,'cat',True] |  |
| d.removeRear() | ['dog',4,'cat',True] | 8.4 |
| d.removeFront() | ['dog',4,'cat'] | True |

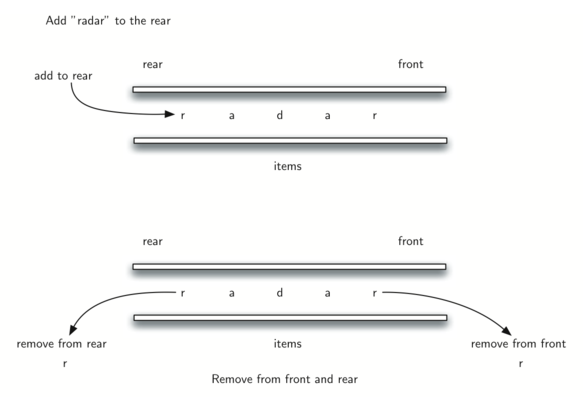
# Palindrome-Checker

## An interesting problem that can be easily solved using the deque data structure is the classic palindrome problem. A palindrome is a string that reads the same forward and backward, for example, radar, toot, and madam.

## Find out words from below which are not palindrome

## IMG_256

## However, we can now make use of the dual functionality of the deque. The front of the deque will hold the first character of the string and the rear of the deque will hold the last character.



# Lists

## A list is a collection of items where each item holds a relative position with respect to the others.



## More specifically, we will refer to this type of list as an unordered list. We can consider the list as having a first item, a second item, a third item, and so on.

## For example, the collection of integers 54, 26, 93, 17, 77, and 31 might represent a simple unordered list of exam scores.

# The Unordered List Abstract Data Type

## *List()* creates a new list that is empty. It needs no parameters and returns an empty list.

## *add(item)* adds a new item to the list. It needs the item and returns nothing. Assume the item is not already in the list.

## *remove(item)* removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.

## *search(item)* searches for the item in the list. It needs the item and returns a boolean value.

## *isEmpty()* tests to see whether the list is empty. It needs no parameters and returns a boolean value.

## *size()* returns the number of items in the list. It needs no parameters and returns an integer.

## *append(item)* adds a new item to the end of the list making it the last item in the collection. It needs the item and returns nothing. Assume the item is not already in the list.

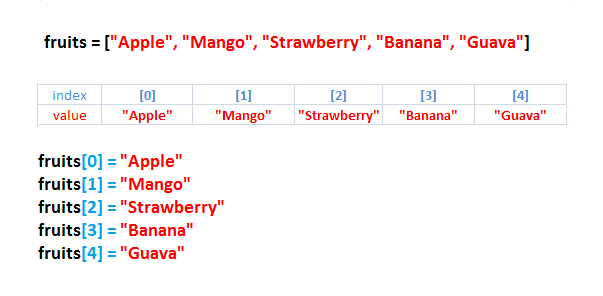
## index(item) returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list

## *insert(pos,item)* adds a new item to the list at position pos. It needs the item and returns nothing. Assume the item is not already in the list and there are enough existing items to have position pos.

## *pop()* removes and returns the last item in the list. It needs nothing and returns an item. Assume the list has at least one item.

## *pop(pos)* removes and returns the item at position pos. It needs the position and returns the item. Assume the item is in the list.

## Example of lists



## You can apply all the list operations mentioned above to this fruits list

## Suppose

## *Fruits.add(‘Orange’)*

## *Fruits.remove(‘Apple)*

## *Fruits.pop()*

## Try all of this by your own. Use python shell or any text editor to perform this.